Army Research Laboratory

A Revised Interface for the ARL Topodef Mobility Design Tool

by Andrew J. Toth and Michael Christensen

ARL-TR-5980 April 2012

Army Research Laboratory
Adelphi, MD 20783-1197

ARL-TR-5980 **April 2012**

A Revised Interface for the ARL Topodef Mobility Design Tool

Andrew J. Toth and Michael Christensen
Computational and Information Sciences Directorate, ARL

REPORT DOCUMENTATION PAGE	*Form Approved* *OMB No. 0704-0188*

1. REPORT DATE *(DD-MM-YYYY)* April 2012	**2. REPORT TYPE** Summary	**3. DATES COVERED (From - To)** October 2010 to September 2011

4. TITLE AND SUBTITLE A Revised Interface for the ARL Topodef Mobility Design Tool	**5a. CONTRACT NUMBER**
	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S) Andrew J. Toth and Michael Christensen	**5d. PROJECT NUMBER**
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) U.S. Army Research Laboratory ATTN: RDRL-CIN-T 2800 Powder Mill Road Adelphi, MD 20783-1197	**8. PERFORMING ORGANIZATION REPORT NUMBER** ARL-TR-5980
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	**10. SPONSOR/MONITOR'S ACRONYM(S)**
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT
Mobile ad-hoc networks provide a means of communicating in areas where there is no established infrastructure. Network emulation has proved to be a good compromise between the cost of full-scale experimentation and the speed of simulation as a means of testing new network protocols and the effects of applications on the network. In order to perform emulation experiments that include mobility, researchers need a tool to design scenarios that closely mimic real-world movement patterns. Scenario generation in the existing tool, Topodef, has proven to be a time-consuming process. Furthermore, maintenance of Topodef has been difficult due to significant coupling between its classes. We set out to create a new Topodef that would speed up the scenario design and generation process for the researcher, as well as improve maintainability and extensibility of the program. To accomplish this, the software was redesigned from the ground up, using the Model-View-Controller design pattern to reduce code coupling. The new user interface borrows elements from the real-time strategy genre of computer games to provide researchers with a path-based input mechanism.

15. SUBJECT TERMS
MANET, emulation, mobility design

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON** Andrew J. Toth
a. REPORT Unclassified	**b. ABSTRACT** Unclassified	**c. THIS PAGE** Unclassified	UU	22	**19b. TELEPHONE NUMBER** (*Include area code*) (301) 394-2746

Standard Form 298 (Rev. 8/98)
Prescribed by ANSI Std. Z39.18

Contents

List of Figures

1. Introduction

Mobile ad-hoc networks (MANETs) are a topic of interest due to the lack of a permanent telecommunications infrastructure in some locations where communication is essential. In a MANET, nodes join and leave the communications network dynamically as they enter and leave the range of a node already within the network. MANET emulation provides a repeatable, scalable environment to assess the impact of network protocols, application software, and environmental effects on network performance. Network simulations lack the fidelity of emulation and do not run actual application and network stack software, while field exercises using production radios are often prohibitively expensive, time consuming, or difficult to coordinate.

A key to developing valuable MANET emulation experiments is defining a realistic mobility scenario. Random models, such as random walk and random waypoint, are simple to implement and analyze, but they do not represent realistic node movements. Researchers need a tool for developing mobility scenarios based on typical troop movement patterns.

Researchers use the U.S. Army Research Laboratory (ARL) Topodef tool (*1*) to define mobility scenarios for MANET experiments. Topodef is a graphical tool for custom-defining a network topology. The original implementation was written in the Python programming language and uses Tkinter to create graphical user interfaces (GUIs) and display animated scenarios (*1*). The tool benefits network science by speeding the creation of new scenarios and editing of existing scenarios that can be used in a variety of MANET experiments.

2. Design Goals

Although the original Topodef application improved the scenario design process from previous tools, several researchers noted room for additional improvement. Editing a scenario required the researcher to position nodes for each time step of the scenario by clicking on the nodes and individually dragging them to the desired positions. The time required to edit a scenario quickly increased with the number of nodes and the length of the scenario. This was the primary motivation for redesigning the Topodef tool.

In addition to improving usability, the new Topodef GUI needed to be more maintainable than the previous version. The original software had a high degree of internal coupling, making modifications and extension extremely difficult. Changes to class files usually meant having to update others. The new Topodef GUI was designed with modularity in mind, allowing for the addition or replacement of individual pieces of code, with minimal restructuring. For example, the ability to have different graphical interfaces for the tool are facilitated by divorcing the GUI code from the underlying objects and data structures using the Model-View-Controller (MVC) design pattern.

3. Inspiration

The core concepts for the controls of the new Topodef GUI were inspired by computer games, namely, games in the real-time strategy (RTS) genre. In this genre, the player controls large numbers of individual units and must command those units to execute arbitrarily complex tactics to accomplish some in-game objective (e.g., destroy the opponent’s army). Decades of trial and error have fine-tuned the controls for these games into a fairly consistent pattern: the player selects units (individually or in a group) with a left mouse click, and then gives the selected unit(s) a destination by clicking again on the main screen area. If the player wishes, they can also queue up a number of waypoints for the selected unit(s), where the unit progresses from their current location to the first waypoint, then the second, and so on. Additional controls are afforded to the player through buttons on the periphery of the screen, which allow the player to tell the selected unit to perform more complex, context-sensitive actions. As a convenience, these action buttons usually have an assigned key on the keyboard so they can be invoked without forcing the player to mouse over away from the main screen area and click them. Figure 1 shows a screenshot from the RTS game *Medieval II: Total War* (*2*).

Figure 1. In *Medieval II: Total War*, battles are fought in an RTS environment, where the player must control hundreds of units at once (*2*).

The new Topodef borrows elements from the RTS control pattern (*3*) because these elements allow for a high degree of control (so that players may execute strategies that require precise movement) while keeping the number of actions to a minimum (allowing the player to execute more commands in a given amount of time). This minimalistic yet powerful input pattern has the added benefit of being intuitive, especially to anyone who has some experience with computer games. Given the application is meant for research use, we placed a higher value on precision and flexibility than the ability to perform actions quickly.

4. Design

After we decided on the general approach to the user interface, we compiled and prioritized a list of features for the application. At the highest priority level were "key features," such as linear node movement, nonlinear node movement, creating new nodes, adjusting existing paths, and executing the defined scenarios (i.e., to have the nodes move along their paths). The next priority level was "non-essential features," such as deleting nodes and paths, adjusting the amount of time to travel along a path, having nodes wait in one place for a set amount of time, and finally, closing a path circuit on which nodes would circle indefinitely. The final category included features was could be included in later versions of the GUI. Ideally, the final program would be easy to extend and modify as new features were needed. This final list was compiled in order to influence the design strategy, i.e., so we would keep in mind that a goal was to be able to add the features without the need for excessive code restructuring. Some examples from this list are dynamic pathing (where nodes would create paths on the fly, reacting to or following the nodes around them), an undo/redo feature, and defining paths for multiple nodes at once. Given these features, a mockup was created to visually demonstrate the most basic functionality of the planned tool (figure 2).

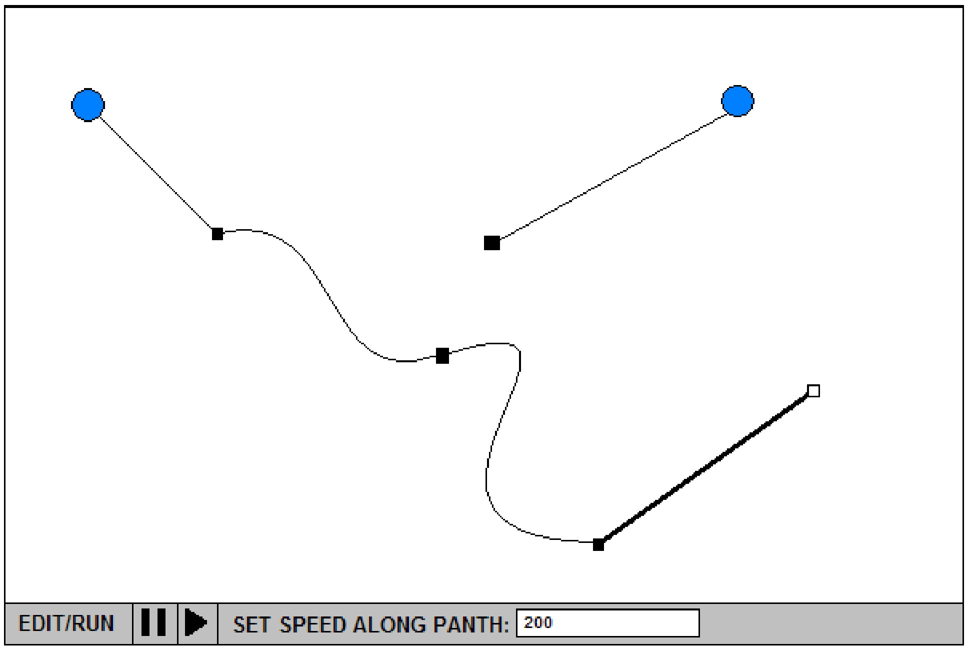


Figure 2. An initial mockup of the Topodef user interface.

The key innovation resulting from the design process is an emphasis on paths—nodes move along researcher-designed paths as though moving down a conveyor belt. Giving paths an existence independent of the nodes that travel along them not only makes their creation easier, but also allows nodes to be added to a path retroactively, as well as allows multiple paths to merge into one. This facilitates the rapid definition of complex scenarios (e.g., creating "patrol" circuits that loop indefinitely) and has the added benefit of easy visualization of node movement even before the scenario is running.

With the functional design idea in mind, it was possible to begin designing the class structure. We restructured the new application from the ground up implementing the MVC design pattern (*4, 5*). The goal was to decouple the user interface from the business model classes. This decoupled approach would provide a consistent application program interface (API) regardless of the user interface. Whether the view is a GUI or a command-line interface, it can depend on the controller to handle the basic operations of object creation, maintenance, traversal, and deletion, without giving it any information about itself.

Figure 3 illustrates the classes required to implement the revised Topodef GUI. This class diagram does not list classes implementing other features of Topodef, such as path-loss models, file import and export, and terrain models. The reasons for this are detailed in section 10.

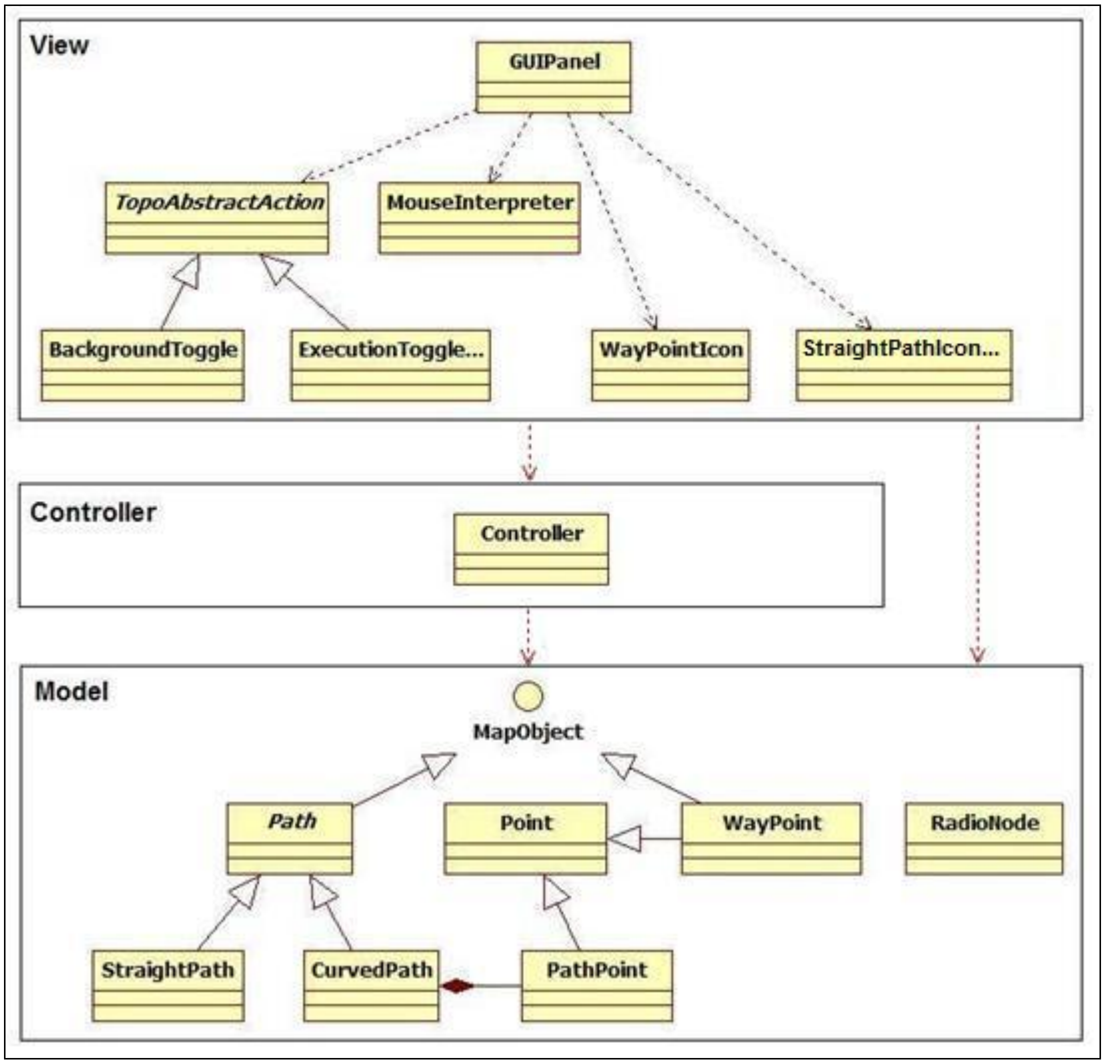


Figure 3. Topodef class model.

5. Model

The code in the model defines the objects that are at the core of the functionality of the software. The RadioNode object fills the roles of the Node object in the existing Topodef application; each instance represents an entity in the field holding one or more radios. These are the objects that actually move around when a scenario is executed. The MapObject class is the base template for objects that deal with directing the RadioNodes. RadioNodes are the only element in the model (besides Point) that is not an extension of MapObject. RadioNodes are placed on MapObjects, and during the execution of a scenario, the MapObject a node is currently "riding on" will tell the node where to go for the next update interval.

Path is an abstract class that defines some functions and constructors that should be common to all Path objects (namely, functions that get or set the WayPoint at the beginning or end of the path instance). The instantiable versions of Path come in two flavors: straight and curved. A StraightPath calculates a dx and dy upon creation, which are used to update any nodes on it by those amounts every frame of animation. CurvedPaths are slightly more complex. In order to simulate smooth curves, CurvedPaths are actually comprised of a series of small line segments between PathPoints; in the current GUI implementation, PathPoints are created every time the mouse listener registers the mouse has been dragged to a new location when the user is in the Path Creation state. In essence, CurvedPaths are just a linked list of PathPoints, and travel along one involves hoping from one PathPoint to the next until the correct distance has been traveled for that time interval.

A special Point class was written for the new Topodef. This may seem redundant due to the existing Point class in Java's libraries, but the purpose of the new Point class was primarily to allow for a shift to three-dimensional modeling (Java's built in point only tracks two dimensions). Additionally the Point class in this project contains a longitude and latitude instead of an *x* and *y* coordinate. Point serves as the superclass for both the WayPoint and PathPoint classes. There was some debate during the design process regarding whether these subclasses should extend the Point class, or merely contain and instance of a Point object. Extension was chosen because on a conceptual level, WayPoint and PathPoint at their most basic level are global positioning system (GPS) coordinates. It made the most sense conceptually to have these classes extend Point because they *are* Points.

6. Controller

The controller maintains the data structures that are holding all of the currently active model members. Its role is to translate commands from the View into actions that the Model will perform. The controller has methods that can be called by a View that will modify a member of, add to, or remove a member of the data structure holding the MapObjects and RadioNodes. This role in the chain of command allows for the View to focus on dealing with user input/output (I/O).

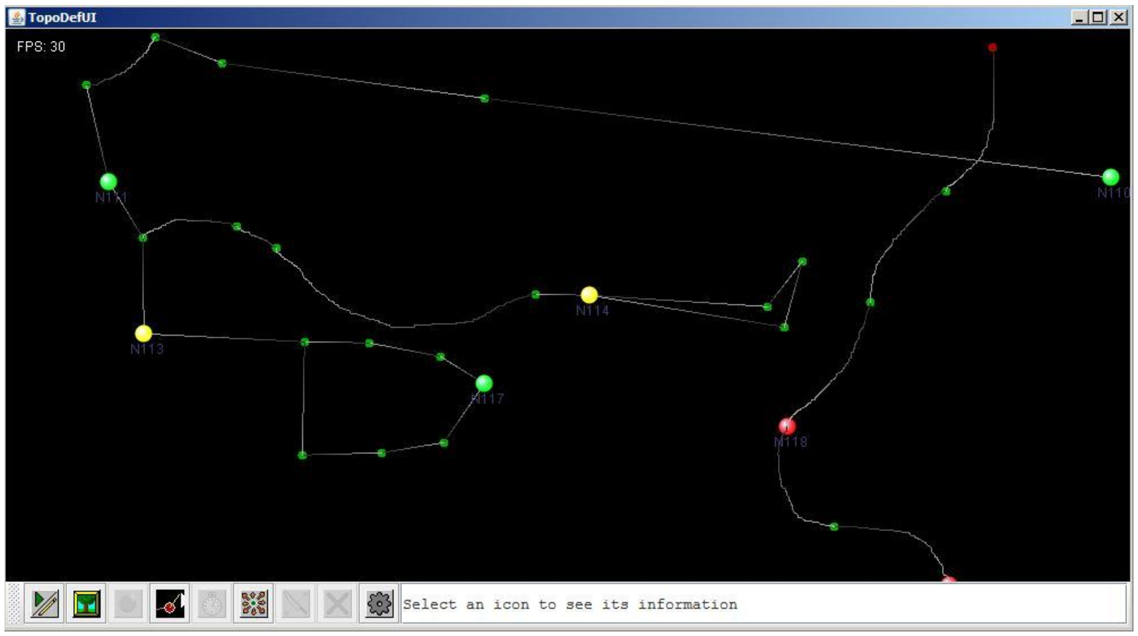
7. View

The view receives user input and relays that input to the controller. The view also reflects changes in the model state as meaningful output in the user display. The current view implementation creates a graphical panel with which the user interacts and has custom buttons-actions and input-panels to provide a streamlined interface. We have designed it with extensibility in mind, relying heavily on Java reflection to allow for the quick addition of new buttons and MapObjects. For example, in order to add a new button to the toolbar, developers create small class file (that defines an icon location, tooltip, and method to call upon the button being activated), and then adds the name of the class to a String array in the main View class.

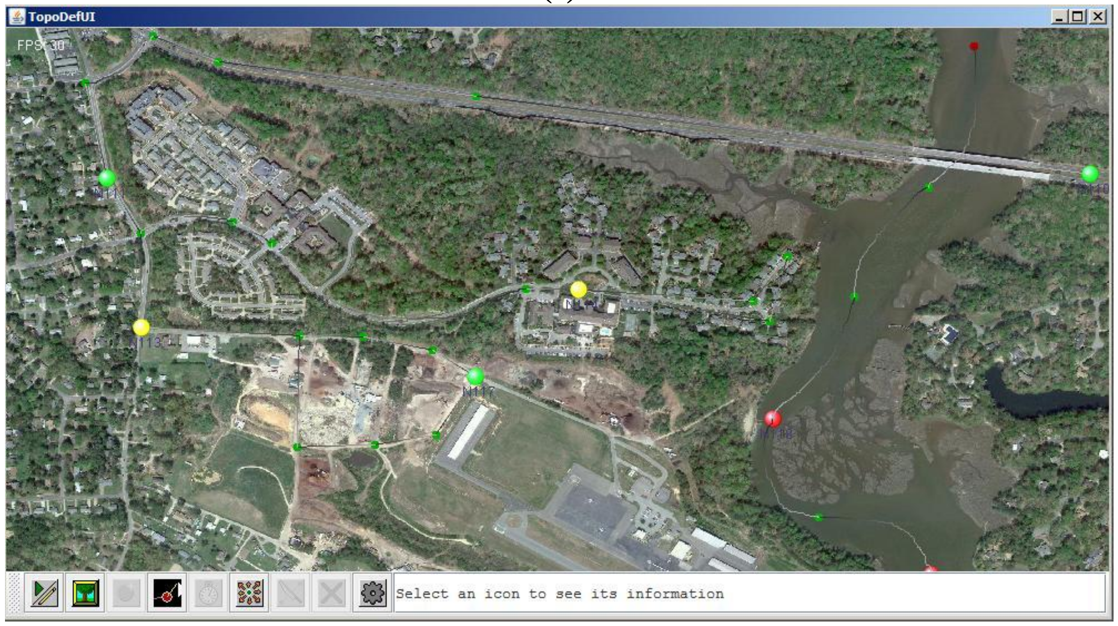
8. Results

The resulting product in its current working state is shown in figure 4a and b. The small circles represent waypoints, with the gradient lines between them representing paths. The gradient on the paths indicates which direction nodes will move when on that path (nodes move from light to dark). The larger labeled spheres represent radio nodes, with each color representing a different platform type (such as tank, high mobility multipurpose wheeled vehicle [HMMWV], or dismount). The representation of nodes and waypoints can be any image file or custom color selected by the user. Below the display area is the toolbar, which contains the buttons used for scenario editing and playback control. From left to right, the components of the toolbar are the edit/execute toggle, background toggle, create nodes button, create path button, edit path button, create randomly moving nodes button, split path button, delete button, configuration button, and the information display. The toolbar is described in more detail in section 9. Actions that require additional information (e.g., creating nodes) will prompt the user with a dialog box when their associated button is activated. Most buttons have an associated hotkey to speed the entry of commands for more experienced users. In order to aid in new users understanding of how their input affects the model, almost all actions are accompanied with one or more of the following:

- a preview of what will happen is presented if the user clicks the mouse in the main panel,
- a dialog box appears after a button is pressed requesting more information from the user before any action is taken, or
- text is written to the display on the toolbar explicitly stating what action was just performed.



(a)

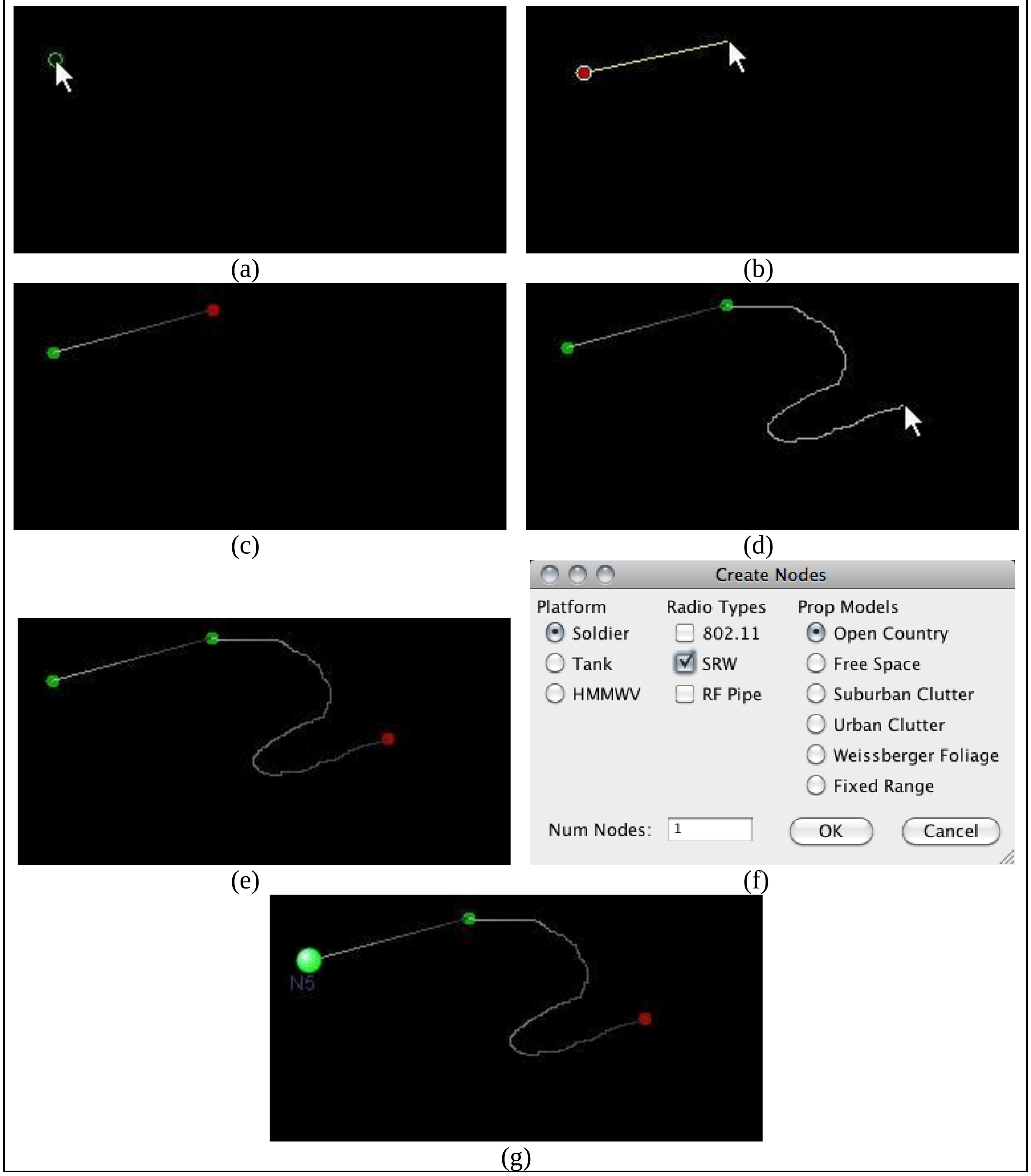


(b)

Figure 4. (a) Map image with node overlay in Topodef GUI and (b) nodes and paths in Topodef GUI.

The new Topodef GUI allows for quicker generation of topology scenarios than its predecessor. The example in figure 4 was created in a couple of minutes, despite advanced node behavior spanning hundreds of time steps. Creating an analogous scenario in the current version of Topodef would take several hours.

Path creation was made to be simple and robust. Users enter the path creation state by pressing the corresponding button on the toolbar. If no waypoint is selected, the user is prompted to place a waypoint with a hollowed-out icon (figure 5a). If a waypoint is selected, upon entering the path creation state the user can immediate place a path leading away from that waypoint. A yellow line is drawn between the waypoint and the cursor showing the user the path that will be created if they click the mouse (figure 5b). By clicking, the user creates a waypoint at the location of the cursor, and then a path is placed between the two points (figure 5c).



Create Nodes

Platform: [✓ Soldier] Tank HMMWV

Radio Types: 802.11 [✓ SRW] RF Pipe

Prop Models: [✓ Open Country] Free Space Suburban Clutter Urban Clutter Weissberger Foliage Fixed Range

Num Nodes: 1

OK Cancel

Figures 5. (a) Placing a waypoint, (b) creating a straight path, (c) a created path, (d) creating a curved path, (e) a created curved path, (f) the create nodes dialog, and (g) a created node.

Alternatively, after entering the path creation mode, the user can click on the selected waypoint, and—while holding the mouse button—drag the mouse to draw a custom curved path (figure 5d). By releasing the mouse button, the path is ended and a waypoint is placed (figure 5e). After creating either kind of path, the user remains in the path creation state, allowing for rapid extension of paths.

Users can place nodes on a waypoint by selecting the waypoint, then pressing the create node button. They are then presented with the dialog shown in figure 5f. This dialog box prompts the user to enter the kind of platform the node will represent, the kinds of radios the node will hold, what kind of propagation model the radio will operate within, and how many nodes to create with the entered specifications. When the user hits OK, the node with the entered properties is placed on the selected waypoint (figure 5g).

If the user ends a path (clicking for a straight path or releasing the mouse button for a curved one) on another waypoint, a merger is created. During execution, the node will travel along the newly created path, and upon reaching the end waypoint, will continue on the path leading away from that waypoint. This intuitive feature allows for the creation of circuits that will execute until the end of the scenario or the merging of two or more paths into one.

At any point during editing, a waypoint can be moved by simply clicking on it, then holding and dragging the mouse. The waypoint and connected paths update in real time as the user makes adjustments, so that the user can see the affect of the change immediately. Waypoints with nodes on them can also be moved with no issue. It must be noted that waypoints cannot be moved if the user is creating a path or attempting to place a new node, since clicking on a waypoint in these cases will result in different actions.

9. Toolbar

The new version of Topodef is implemented using the Java programming language and includes a toolbar at the bottom of the main window. A notable feature of the Java toolbar widget is that it can be dragged away from the frame containing it and exist independent of the frame. Toolbars may also be repositioned along any other border of the program’s frame, but it is recommended to only bind the toolbar to the top or bottom of the frame due to the horizontal text display.

9.1 Execution/Edit Toggle (E)

This button starts and stops execution of the designed scenario. If the program is in execution mode, no changes can be made to the scenario and the nodes move along their paths until they reach a waypoint that doesn’t have a path leading away. Execution can be started at anytime during editing and can be stopped at any point during the playback as well. The hotkey for this button is E.

9.2 Background Toggle (B)

This button essentially lays an image (named “map.png” in the “images” package) to be used as a background instead of the normal black or gray. The advantage is scenarios can be created to follow real-world geography and take on a more definite relevance. The advantage of leaving the background off is it is much easier to see the already created paths and waypoints against the darker, uniform backgrounds. The background can be toggled on or off at any point of editing or execution. The hotkey for this button is B.

9.3 New Node (N)

The action associated with this button behaves differently based on the current state of the program. If nothing or a path is selected, pressing this button or hitting its associated hotkey will allow the user to place a new node anywhere on the screen. This is accomplished by first creating a waypoint at the location selected, then placing a node on top of that waypoint. Once the waypoint and node have been placed, the state of the program switches to path creation; creating a new node in this manner is the simplest way to create new paths. It is also worth noting that the user is clued into this functionality via a hollowed waypoint icon following the mouse cursor around the main panel.

The action performed is different if a waypoint is selected when the button or hotkey is pressed. In this situation, a dialog box prompts the user to enter the number of nodes they would like to add to this waypoint. This feature allows researchers to place multiple nodes on top of one point, so they travel exactly together (simulating multiple radios in the same vehicle, for example). The default value for this dialog box is 0, so that if the user mistakenly brought it up, it can be removed quickly without any repercussions in the model. The hotkey for this button is N.

9.4 Create Path (P)

If the user has selected a waypoint without a path leading away from it (denoted by its being red), then they can trigger this action to begin adding new paths. As mentioned before, once the user enters the path creation state, a preview of the straight path is drawn between the selected waypoint and the mouse pointer. A click will result in the demonstrated path, whereas a click and drag originating from the selected waypoint will result in a curved path exactly mimicking the user’s mouse movements up until the point of release. If already in this path creation state, triggering the action again will return the user to the generic editing state. The hotkey for this button is P.

9.5 Edit Time (T)

This action allows users to change the amount of time to traverse a path or wait at a waypoint. To accomplish this, the user just has to select the object of interest, and then trigger this action.

A dialog box will appear prompting for the new amount of time to traverse the selected object (to be input in milliseconds). The default value for this input box is the current amount of time for traversal, so any accidental trigger of the edit time action is less likely to adversely affect the model. They hotkey for this button is T.

9.6 Delete Selected (Del)

The final button on the toolbar, this button removes the selected object from the model. The current implementation will start by removing the selected object, and then recursively remove any waypoints and paths that proceed from the deleted piece. This recursive deleting stops once the end of a path is reached, a waypoint is found on the path that has a node sitting on it, or a waypoint is found that has multiple incoming paths. This recursive algorithm was implemented because any path that no longer had a node that would traverse it becomes irrelevant. The hotkey for this button is the Delete key.

9.7 Text Display

The text display shows contextually appropriate information to the user. Its default output is to show information about the currently selected object (time to traverse, position, etc.), but it is also used to explain input formatting errors or explicitly state what action the user has just performed.

9.8 Escape Key

The escape key is also used to perform cancellation actions. If the user is creating a new node or a new path and wishes to return to the generic editing mode, the user can press the universal escape in addition to the dedicated path/node actions. The escape key will also stop an executing scenario and return to editing.

10. Future Work

The Topodef GUI is a work in progress. Some of the features currently in development are the ability to save and load scenarios, and generate mobility model data files for use in network science experiments. These features will greatly enhance the software's usefulness to researchers.

The effort to redesign the scenario editing portion of Topodef was done without the constraints imposed by the existing software in an effort to explore as many options as possible. For that reason, the GUI described in this report does not integrate with the existing Topodef application. A separate effort is currently underway to redesign the architecture of Topodef to integrate it with the GUI updates and to improve performance of path-loss calculations used to visualize network topologies. Another effort underway is the integration of NetDMF, which is a mechanism for storing large data sets developed as a part of the Mobile Network Modeling Institute (*6*). The revised Topodef will also include support for the automation efforts currently underway in ARL's Wireless Emulation Laboratory.

11. References

1. Nguyen, B. *The ARL Topodef Tool—Concept and Usage*; ARL-TR-5002; U.S. Army Research Laboratory: Adelph, MD, October 2009.

2. Medieval II: Total War Gallery [Screenshot gallery]. (n.d.). Retrieved from http://media.gamedaily.com/games/medieval-ii-total-war/pc/tn_565_1.jpg

3. Real-Time Strategy – Wikia Gaming, http://gaming.wikia.com/wiki/Real-time_strategy

4. *Java SE Application Design with MVC* [Technical article]. (2010). Retrieved from Oracle Corporation, website: http://java.sun.com/developer/technicalArticles/javase/mvc

5. *Java BluePrints Model-View-Controller* [Pattern blue print]. (2002). Retrieved from Sun Microsystems, website: http://java.sun.com/blueprints/patterns/MVC-detailed.html

6. Mobile Network Modeling Institute, https://hsai-web.arl.army.mil/pub/mnmi/

List of Symbols, Abbreviations, and Acronyms

API	application program interface
ARL	U.S. Army Research Laboratory
GPS	global positioning system
GUI	graphical user interface
HMMWV	high mobility multipurpose wheeled vehicle
I/O	input/output
MANETs	mobile ad-hoc networks
MVC	Model-View-Controller
RTS	real-time strategy

NO. OF COPIES	ORGANIZATION
1 (PDF ONLY)	DEFENSE TECHNICAL INFORMATION CTR DTIC OCA 8725 JOHN J KINGMAN RD STE 0944 FORT BELVOIR VA 22060-6218
1	DIRECTOR US ARMY RESEARCH LAB IMNE ALC HRR 2800 POWDER MILL RD ADELPHI MD 20783-1197
1	DIRECTOR US ARMY RESEARCH LAB RDRL CIO LL 2800 POWDER MILL RD ADELPHI MD 20783-1197
1	DIRECTOR US ARMY RESEARCH LAB RDRL CIO LT 2800 POWDER MILL RD ADELPHI MD 20783-1197
1	DIRECTOR US ARMY RESEARCH LAB RDRL D 2800 POWDER MILL RD ADELPHI MD 20783-1197
2	US ARMY RSRCH LAB ATTN RDRL CIN T ANDREW TOTH MICHAEL CHRISTENSEN 2800 POWDER MILL RD ADELPHI MD 20783-1197